8-23-06

IFW

PTO/SB/21 (07-06)
Approved for use through 09/30/2006. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	10/616,081-Conf. #9680
Filing Date	July 8, 2003
First Named Inventor	Karine Excoffier
Art Unit	2163
Examiner Name	M. R. Filipczyk
Attorney Docket Number	03226/501001; P7646
Total Number of Pages in This Submission	

ENCLOSURES (*Check all that apply*)

- □ Fee Transmittal Form
  - □ Fee Attached
- □ Amendment/Reply
  - □ After Final
  - □ Affidavits/declaration(s)
- □ Extension of Time Request
- □ Express Abandonment Request
- □ Information Disclosure Statement
- ☒ Certified Copy of Priority Document(s) (25 pages)
- □ Reply to Missing Parts/ Incomplete Application
  - □ Reply to Missing Parts under 37 CFR 1.52 or 1.53

- □ Drawing(s)
- □ Licensing-related Papers
- □ Petition
- □ Petition to Convert to a Provisional Application
- □ Power of Attorney, Revocation Change of Correspondence Address
- □ Terminal Disclaimer
- □ Request for Refund
- □ CD, Number of CD(s) ______
  - □ Landscape Table on CD

- □ After Allowance Communication to TC
- □ Appeal Communication to Board of Appeals and Interferences
- □ Appeal Communication to TC **(Appeal Notice, Brief, Reply Brief)**
- □ Proprietary Information
- □ Status Letter
- ☒ Other Enclosure(s) (please Identify below):

Return Receipt Postcard
Submission of Priority Documents Under 35 U.S.C. 119 (2 pages)

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm Name	OSHA · LIANG LLP		
Signature			
Printed name	Robert P. Lord T. Chyau Liang #48,885		
Date	August 22, 2006	Reg. No.	46,479

Docket No.: 03226/501001; P7646
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Karine Excoffier et al.

Application No.: 10/616,081

Confirmation No.: 9680

Filed: July 8, 2003

Art Unit: 2163

For: INDEXING VIRTUAL ATTRIBUTES IN A DIRECTORY SERVER SYSTEM

Examiner: M. R. Filipczyk

SUBMISSION OF PRIORITY DOCUMENTS UNDER 35 U.S.C. 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Concurrent with the filing of the above referenced patent application, Applicants claimed priority under 35 U.S.C. 119 to the following prior foreign application filed in the following foreign country on the date indicated:

Country	Application No.	Date
France	FR0208558	July 8, 2002

This priority claim was acknowledged on the filing receipt mailed October 7, 2003. In support of this claim, a certified copy of the original foreign application is filed herewith.

Applicant believes no fee is due with this response. However, if a fee is due, please charge our Deposit Account No. 50-0591, under Order No. 03226/501001.

Dated: August 22, 2006

Respectfully submitted,

By
for Robert P. Lord T. Chyau Liang
Registration No.: 46,479 #48,885
OSHA · LIANG LLP
1221 McKinney St., Suite 2800
Houston, Texas 77010
(713) 228-8600
(713) 228-8778 (Fax)

Application No. (if known): 10/616,081 Attorney Docket No.: 03226/501001; P7646

Certificate of Express Mailing Under 37 CFR 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail, Airbill No. EV 804198946 US in an envelope addressed to:

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

on August 22, 2006 .
Date

Signature

Mona F. Hernandez
Typed or printed name of person signing Certificate

Registration Number, if applicable

(713) 228-8600
Telephone Number

Note: Each paper must have its own certificate of mailing, or this certificate must identify each submitted paper.

Transmittal (1 page)
Claim for Priority and Submission of Documents (2 pages)
Certified Copy of French Patent Application No. FR0208558 filed 7/8/2002 (25 pages)

REPUBLIQUE FRANÇAISE

INPI
INSTITUT
NATIONAL DE
LA PROPRIETE
INDUSTRIELLE

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 10 AOUT 2006

Pour le Directeur général de l'Institut
national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

10/616,081

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

cerfa N° 11354*01

REQUÊTE EN DÉLIVRANCE 1/2

Important ! Remplir impérativement la 2ème page.

Cet imprimé est à remplir lisiblement à l'encre noire DB 540 W /190600

Réservé à l'INPI	**1** NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE
REMISE DES PIÈCES DATE 8 JUIL 2002 LIEU 75 INPI PARIS N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0208558 DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI 0 8 JUIL. 2002	CABINET NETTER 36 avenue Hoche 75008 PARIS
Vos références pour ce dossier *(facultatif)* SUN 40 (120720)	

Confirmation d'un dépôt par télécopie □ N° attribué par l'INPI à la télécopie

2 NATURE DE LA DEMANDE	**Cochez l'une des 4 cases suivantes**		
Demande de brevet	☒		
Demande de certificat d'utilité	□		
Demande divisionnaire	□		
Demande de brevet initiale	N°	Date	/ /
ou demande de certificat d'utilité initiale	N°	Date	/ /
Transformation d'une demande de brevet européen *Demande de brevet initiale*	□ N°	Date	/ /

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

Indexing virtual attributes in a directory server system.

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE	Pays ou organisation Date / /	N°
	Pays ou organisation Date / /	N°
	Pays ou organisation Date / /	N°
	□ **S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»**	

5 DEMANDEUR		□ **S'il y a d'autres demandeurs, cochez la case et utilisez l'imprimé «Suite»**
Nom ou dénomination sociale		SUN MICROSYSTEMS, INC
Prénoms		
Forme juridique		
N° SIREN		
Code APE-NAF		
Adresse	Rue	901 San Antonio Road
	Code postal et ville	94303 PALO ALTO Californie
Pays		Etats-Unis d'Amérique
Nationalité		Société des Etats-Unis d'Amérique
N° de téléphone *(facultatif)*		
N° de télécopie *(facultatif)*		
Adresse électronique *(facultatif)*		

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

REQUÊTE EN DÉLIVRANCE 2/2

Réservé à l'INPI

REMISE DES PIÈCES
DATE 8 JUIL 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0208558

DB 540 W /190600

Vos références pour ce dossier : *(facultatif)*		SUN Aff. 40 (120720)
6 MANDATAIRE		
Nom		PLAÇAIS
Prénom		Jean-Yves
Cabinet ou Société		Cabinet NETTER
N °de pouvoir permanent et/ou de lien contractuel		
Adresse	Rue	36 avenue Hoche
	Code postal et ville	75008 PARIS
N° de téléphone *(facultatif)*		01 58 36 44 22
N° de télécopie *(facultatif)*		01 42 25 00 45
Adresse électronique *(facultatif)*		
7 INVENTEUR (S)		
Les inventeurs sont les demandeurs		□ Oui ☒ Non **Dans ce cas fournir une désignation d'inventeur(s) séparée**
8 RAPPORT DE RECHERCHE		**Uniquement pour une demande de brevet (y compris division et transformation)**
Établissement immédiat ou établissement différé		□ ☒
Paiement échelonné de la redevance		**Paiement en deux versements, uniquement pour les personnes physiques** □ Oui □ Non
9 RÉDUCTION DU TAUX DES REDEVANCES		**Uniquement pour les personnes physiques** □ Requise pour la première fois pour cette invention *(joindre un avis de non-imposition)* □ Requise antérieurement à ce dépôt *(joindre une copie de la décision d'admission pour cette invention ou indiquer sa référence) :*
Si vous avez utilisé l'imprimé «Suite», indiquez le nombre de pages jointes		

10 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire) N° Conseil 92-1197 (B) (M) Jean-Yves PLAÇAIS	**VISA DE LA PRÉFECTURE OU DE L'INPI** C. MARTIN

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire. Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

Indexing virtual attributes in a directory server system

This invention relates to distributed computer systems.

In certain fields of technology, complete computer systems, including a diversity of equipments, are optimized for storing and retrieving data. Such systems may provide services to user machines related to a local network, e.g., an Intranet, or to a global network, e.g., the Web network.

It is desirable that network users can access, upon a query, to a large number of data, making it possible for the network users to create their own dynamic web site or to consult a dynamic web site, for example an e-commerce site on a multi platform computer system (Solaris, Windows NT). These queries are directed to a directory, e.g., a LDAP directory, and managed by a directory server. It is further desirable that this access to a large number of data be made possible more rapidly for each query arriving after a first query.

A general aim of the present invention is to provide advances in these directions.

Broadly, there is proposed a directory server, capable of interacting with entries organized in a tree structure. Each entry has attributes, these attributes comprising real attributes each having a value stored in the entry. The directory server comprising:
- a mechanism capable of associating a virtual attribute to an entry, subject to a virtual attribute condition being verified, the virtual attribute condition being derived from data located elsewhere in the tree structure, and
- a resolving function, capable of receiving a first filter expression, based on a virtual attribute, for converting it into one or more second filter expressions, containing real attributes, and being computed from the first filter expression and from the virtual attribute condition.

There is also proposed a method of operating a directory server system, comprising a directory server interacting with entries organized in a tree structure, each entry having

α

attributes. The attributes comprise real attributes, each real attribute having a value stored in the entry, and virtual attributes, each virtual attribute being associated to an entry, subject to a virtual attribute condition being verified. The virtual attribute condition is derived from data located elsewhere in the tree structure. The method comprises the steps of :

a) receiving a first filter expression,

b) if the first filter expression is based on a virtual attribute, converting said first filter expression into one or more second filter expressions, containing real attributes, said one or more second filter expressions being computed from the first filter expression and from the virtual attribute condition.

This invention may also be defined as an apparatus or system, and/or as software code for implementing the method, or for use in the system, in all their alternative embodiments to be described hereinafter.

Other alternative features and advantages of the invention will appear in the detailed description below and in the appended drawings, in which :

- Figure 1 is a general diagram of a computer system in which the invention is applicable;
- Figure 2 illustrates a typical LDAP exchange between a LDAP client and a LDAP server, and between the LDAP server and further servers;
- Figure 3 illustrates the general structure of a LDAP directory;
- Figure 4 shows a portion of a LDAP tree;
- Figure 5 illustrates attribute types and values of a directory entry;
- Figure 6 is a flowchart representing the operations performed for evaluating a search request, according to the prior art;
- Figure 7 illustrates the scope of a role;
- Figure 8 represents the structure of three types of roles, according to the prior art;
- Figure 9a is a flowchart for enumerating the roles possessed by a given entry, according to the prior art;
- Figure 9b is a flowchart for determining whether a given entry is member of an existing role, according to the prior art;
- Figure 10a represents the general structure of a filter execution function according to the embodiments of this invention;

α

- Figure 10b is a flowchart for indexing the virtual attribute *nsrole*, according to an embodiment of this invention;
- Figure 11 represents a portion of a directory tree illustrating a scope restriction;

Additionally, the detailed description is supplemented with the following Exhibits:
- Exhibit E1 contains definitions related to search filters,
- Exhibit E2 contains role definition entries.

In the foregoing specification, references to the Exhibits are made directly by the Exhibit or Exhibit section identifier: for example, E2.1 refers to section E2.1 in Exhibit E2. The Exhibits are placed apart for the purpose of clarifying the detailed description, and of enabling easier reference.

Now, making reference to software entities imposes certain conventions in notation. Particularly, an expression indicated between the quote signs " " may be used to design LDIF extracts and an expression in italics may be used for representing an attribute, an object class or a LDAP operation.

As they may be cited in this specification, Sun, Sun Microsystems and Sun One are trademarks of Sun Microsystems, Inc.

This invention may be implemented in a computer system, or in a network comprising computer systems. Figure 1 represents an example of the hardware of such computer systems. The hardware comprises :
- a processor CPU 11, e.g. an Ultra-Sparc;
- a program memory 12, e.g. an EPROM, a RAM, or Flash memory;
- a working memory 13, e.g. a RAM of any suitable technology;

- a mass memory 14, e.g. one or more hard disks;
- a display 15, e.g. a monitor;
- a user input device 15, e.g. a keyboard and/or a mouse;
- a network interface device 21 connected to a communication medium 20, which is in communication with other computers. Network interface device 21 may be of the type of Ethernet, or of the type of ATM. Medium 20 may be based on wire cables, fiber optics, or radio-communications, for example.

Data may be exchanged between the components of figure 1 through a bus system 10, represented as a single bus for simplification of the drawing. Bus systems may include a processor bus, e.g. PCI, connected via appropriate bridges to, e.g. an ISA or a SCSI bus.

The data exchanged are handled by a resource provider using a server to deliver data to user computers, or to store the data provided by the user computers. Browsers, e.g. Internet Explorer, are further provided on user computers, to enable users to make requests, to retrieve or store data. The resource provider makes it possible for user computers on a network to share data of any kind.

iPlanet E-commerce Solutions, now Sun One E-commerce Solutions, has developed a "net-enabling" platform called the Internet Service Deployment Plateform (ISDP). ISDP includes multiple, integrated layers of software that provide a full set of services supporting application development, e.g., business-to-business exchanges, communications and entertainment vehicles, and retail Web sites.

Sun One™ Directory Server provides a centralized directory service directory service for an intranet or an extranet. A directory service represents a collection of software, hardware, and processes that are able to deliver and store information. The directory service generally includes one or more directory client programs that can access the data stored in the directory, e.g. names, phone numbers or addresses.

The Sun One Directory Server is a general purpose directory that stores all information in a single, network-accessible repository. The Sun One Directory Server provides the standard protocol LDAP and an application programming interface (API) to access the information contained by the Sun One Directory Server.

LDAP is the Internet standard for directory lookups, just as the Simple Mail Transfer Protocol (SMTP) is the Internet Standard for delivering e-mail and the Hypertext Transfer Protocol (HTTP) is the Internet standard for delivering documents. Technically, LDAP is defined as on-the-wire bit protocol (similar to HTTP) that runs over Transmission Control Protocol/ Internet Protocol (TCP/IP). It specifies the interaction between clients and servers and determines how LDAP queries and responses are carried over the IP network.

A LDAP-compliant directory, such as the Sun One Directory Server, leverages a single, master directory that contains all users, groups and access information. The directory is hierarchical, not relational and is particularly fitted for reading while offering a high reliability and a high scalability.

Referring now to figure 2, LDAP defines a communication 1 between a server 17 and a client 18. LDAP also defines a communication 2 between LDAP server 17 and servers 17.1 to 17.n, which makes it possible for the server LDAP 17 to exchange its content (replication service) with servers 17.1 to 17.n or to access the directory of one of the servers 17.1 to 17.n (referral service) and vice versa.

The LDAP protocol is a message-oriented protocol. The client 18 constructs a LDAP message containing a request and sends the message to the server 17. The server 17 processes the request and sends a result, or results, back to the client 18 as a series of LDAP messages.

Such a client-server communication additionally lies on a specific architecture. LDAP creates a standard defining the way data are exchanged between the client computer and the directory server and defining the way data are modeled. More specifically, LDAP relies on four basic models:

- an information model;
- a naming model;
- a functional model; and
- a security model.

α

The LDAP information model defines the kind of data that can be stored in a directory. LDAP directory is populated with entries. An entry corresponds to real-world objects, such as a person, a printer, or configuration parameters.

Figure 3 illustrates the general structure of a LDAP directory : the directory server 30 executes implemented functions based on the entries 31 stored in databases. The entries comprise configuration entries 310, user entries 311 and administrative entries 312. These entries further interact with the schema 32 described below.

The configuration entries are stored under the subtree "cn=config". The user entries comprise data related to the users of the directory server. Administrative entries relate to user management and are generally implemented as LDAP subentries.

An entry contains a set of attributes associated with values. Each entry is uniquely identified by its distinguished name DN. The distinguished name may be stored in the attribute *dn* (*distinguishedName*).

LDAP entries are organized in a hierarchical tree structure, called the Directory Information Tree (DIT). Each node of the tree comprises an entry. Figure 4 illustrates an organization entry (22) with the attribute type of domain component *dc*, an organizational unit entry (24) with the attribute type of organizational unit *ou*, a server application entry (26) with the attribute type of common name *cn*, and a person entry (28) with the attribute type of user ID *uid*. The entries are connected by the directory. Each server has a particular entry called root directory specific entry (rootDSE) which contains the description of the tree and of its content.

A LDAP Data Interchange Format (LDIF) is an ASCII text file format used to describe directory entries and operations on those entries. It enables to create, modify, and delete Directory entries and to import and export data among LDAP directories. Figure 5 is a LDIF representation of an entry 404, showing the attribute types 400 and their values 402.

The information model is extensible, which means that new types of information can be

α

added to a LDAP directory.

Descriptive information is stored in the attributes of the entry. Each attribute describes a specific type of information. Attributes may have constraints that limit the type and length of data placed in attribute values.

All entries require the *objectclass* attribute which lists the object classes to which an entry belongs. An entry can belong to one or more object classes and must satisfy all of them. The *objectclass* attribute defines which attributes are required and which attributes are allowed in the entry.

For example, in figure 5, the entry (404) represented in LDIF belongs to the object classes *top*, *person*, *organizationalPerson* and *inetOrgPerson.*

Each attribute has a corresponding syntax definition. The syntax definition describes the type of information provided by the attribute. The object classes, the required and allowed attributes, and the syntax definition of the attributes are listed in the directory schema.

The LDAP directory comprises a structure 32, represented in figure 3, that defines object classes and attributes, and may be viewed as metadata. This structure, called the schema, sets the rules defining what information can be stored in the LDAP directory and how information is organized. The schema specifies the required and allowed attributes that are used to store information and theirs syntax definition. A schema checking function may be activated, thus causing the directory server to check new entries to verify :

- object classes and attributes attached to new entries are defined in the schema 32,
- the attributes required for an object class according to the schema 32, are contained in an entry attached to that object class,
- only attributes allowed by the object class according to the schema 32, are contained in an entry attached to that object class.

The LDAP naming model specifies that directory entries must be hierarchical and organized in an inverted tree structure. As mentioned above, each entry has a unique name called a distinguished name *dn*. The *dn* consists of the name of a list of the names of all the parent entries in the directory back to the top of the directory hierarchy, the name of the entry being at the extreme left, e.g., "uid=Joe,ou=people,dc=france,dc=sun,dc=com", in figure 5. The root of the entry is at the extreme right of the *dn*. The name at the extreme left of the *dn*, "uid=Joe " in the example, is the relative distinguished name or *rdn*. Within a set of entries sharing the same parents, the *rdn* must be unique. This ensures that two entries in the directory tree cannot have the same *dn*.

The LDAP functional model comprises eight basic functional operations that a user from a client computer can perform on the directory data of a LDAP directory server :

- *bind* and *unbind* : begin and end the exchange of information between LDAP clients and the directory server;
- *add*, *delete*, and *modify* : apply on specific entries in the DIT,
- *compare* : applies on two entries to compare their content according to criteria,
- *search* : locates specific entries in the DIT,
- *modifyRDN* : applies to change the distinguished name *dn* of an entry.

In addition to the eight basic functional operations, the LDAP protocol defines a framework for adding new operations to the protocol via LDAP extended operations. Extended operations allow the protocol to be extended in an orderly manner to meet new marketplace needs as they emerge.

The *search* operation supported by LDAP allows clients to search the directory for data. Search operations are performed by search filters.

A search filter selects entries in the DIT, based on criteria defined in a filter expression. They are mostly used with the *ldapsearch* function. Referring to Exhibit E1.1, a *ldapsearch* function has the following parameters :

- *options*: represents a series of command-line options. They are specified before the search filter;
- *search filter*: represents a LDAP filter expression;
- *list of attributes*: represents a list of attributes separated by a space. This list of attributes is returned after the search filter. If the list of attributes is not specified, the search provides values for all attributes permitted by the access control set in the directory ;

Exhibit E1.2 represents the main options used for performing a ldap search.

A filter expression is the expression of a condition on given attributes. The basic syntax of a filter expression comprises :

- an attribute
- an operator
- a value

For example, in the search filter "PostalCode = 75006 ", "PostalCode " is the attribute, "=" is the operator, and "75006" is the value.

A search filter may further comprise several attributes combined by boolean operators.

The type of a Ldap search filter depends on the operator used. For example, a search filter using the operator "=" is an equality filter. An equality filter, e.g. "sn=Jensen", returns entries that comprise attribute values exactly matching the value specified in the filter.

Exhibit E1.3 contains the definitions of the main search filter types. Additionally, search filter may be combined using boolean operators to form complex expressions. An example of complex filter expression is shown in Exhibit E.1.4.

Exhibit E.1.5 contains the definition of the main boolean operators. In a search filter comprising boolean expressions, the expressions between parentheses are evaluated from left to right, and for each expression, the sub parenthetical expressions are evaluated from the innermost to the outermost. Thus, in the example shown in Exhibit E.1.4, expressions are evaluated in the order

1- (!cn=Bob Jones*)

2- (|(cn=Bob*)(cn=*Jones))

3- (age<=40),

and in (|(cn=Bob*)(cn=*Jones)), expressions are evaluated in the order :

2.1- (cn=Bob*)

2.2- (cn=*Jones)

2.3- (|(cn=Bob*) (cn=*Jones))

When a LDAP search is received by the server comprising a search filter, the search filter is evaluated and divided into elementary search filters as mentioned above.

The performance of a LDAP search may be improved by the use of indexes. Indexes are files stored in the directory databases, which may be cached. Such index files contain indexing tables. Indexing tables maintain a list of the entries matching a given value of an attribute or indexed attribute, according to a given search type. In the indexes, the candidate entries are identified by an identifier such as an ID number.

More specifically, indexes resolve the problem of restricted search scope. Indeed, an efficient search may be performed only if the user can restrict the scope of the search, which requires some knowledge of the directory structure. With no structure knowledge, the search is performed all over the tree structure. Indexes provide a preselection of entries, obtained during a previous search of a given type. Instead of repeating the same search every time it is requested, the directory server stores the result obtained in the previous search in indexes and updates these indexes when an attribute or an attribute value is modified.

The names of the index files are based on the indexed attribute, e.g. *sn.db*. Each index file may contain different types of indexes. A directory server supports the following types of indexes, in accordance with the existing search filter types:

- the *presence index* lists the entries that contain a particular attribute, such as *cn*,
- the *equality index* list the entries that contain a specific attribute value, such as "cn= bob",
- the *approximate index* allows approximate searches, such as "cn~= bob",
- the *substring index* allows searches against substrings within entries, such as "cn=*peters"
- the *browsing index*, or virtual list view index, speeds up the display of entries in the directory server console. This index is particularly useful for the branches of the directory that contain hundreds of entries, for example, the ou=people branch.
- the *international index* speeds searches for information in international directories.

α

Attributes that are not indexed can be specified in filter expressions, although the search performance may be degraded according to the type of search. For such attributes, the directory server only examines a subset of the entries of the DIT. However, maintaining indexes for attributes that are not used in a search may also degrade the search performance.

Indexes are used to speed up searches. Figure 6 is a flowchart representing the operations performed for evaluating a search request, according to the prior art.

At operation 100, the directory server receives a search request from a client. Operation 102 checks whether the base DN specified in the request matches a suffix contained by at least one of its database. If so, the directory processes the request. Otherwise, the directory server returns an error message (operation 103) to the client indicating that the suffix does not match.

Operation 104 breaks down this search filter contained by the request into elementary search filters of the type "attribute operator value". The directory then processes each elementary search filter as follows.

Operation 105 determines whether the attribute mentioned in the elementary search filter is associated with an index, and if so the server reads that index (operation 106) to generate a list of candidate entries potentially matching the request. More exactly, the directory server takes the list of candidate entries from the index as a series of entry ID numbers, and reads the corresponding entries from a file that contains the actual directory database entries.

If the attribute mentioned in the elementary search filter is not associated with an index (test 105), operation 108 generates a candidate list that includes all entries in the database, which makes the search considerably slower.

The directory server repeats operations 105 and 106 for each elementary search filter and when all the elementary search filters are processed (test 109), operation 110 combines the candidate entries obtained for the elementary search filters.

ℴ?

If a search request mentions several attributes (test 109), the directory server checks several indexes and then combines the resulting lists of candidate entries at operation 110.

For each one of the entries of the candidate list provided by operation 110 or 108, the directory server determines whether the entry matches the search criteria (operation 112). If so, at operation 116, the directory server adds the entry to the result.

The directory server stops and transmits the result to the client, at operation 118, when a predefined limit is reached (test 114). The predefined limit may be reached when all the candidate entries have been examined. Alternatively, the limit may be set by one of the following attributes :

- *nsslapd-sizelimit* which sets the maximum number of entries to return from a search operation.
- *nsslapd-timelimit* which sets the maximum number of seconds to spend on a search request.
- *nsslapd-lookthroughlimit* which sets the maximum number of candidate entries to examine during a search request.

Searching is efficiently handled through the use of indexes that are maintained by the directory. However, some attributes cannot be indexed.

In particular, virtual attributes do not support indexing. Virtual attributes are attributes that are not stored in the entry itself but computed according to a condition derived from data stored elsewhere in the directory. The most commonly used are *nsrole* attribute and *CoS* attribute. In particular, the virtual attribute *nsrole* is computed to indicate all the roles possessed by a user entry and a *CoS* attribute is generated by class of service (CoS) in target entries to allow these entries to share a same value of the *CoS* attribute without having to store it in each entry.

Virtual attributes are not indexed using the existing indexing methods, because updating indexes based on such attributes involve bad performances. Indeed, a change made on the definition of a virtual attribute, e.g. on a role definition or on a CoS definition, would require the regeneration of the entire virtual attribute index, which would be too costly.

a

More generally, a LDAP directory server does not support LDAP search requests containing a filter that references virtual attributes. A search filter based on a virtual attribute, e.g. "nsrole = cn=ExampleRole, ou = people, dc = example, dc = com", may return erroneous results when executed.

Only attributes stored in entries (or real attributes) are efficiently supported in LDAP search filters. To search entries, based on the values of a virtual attribute, a directory client must retrieve a set of the entries, such as an entire branch, and sort through them to select the entries of interest.

The invention addresses the above problems.

The foregoing description refers to *nsrole* virtual attribute, as an exemplary application of this invention.

According to the prior art, *nsrole* attribute is a multi-valued attribute that indicates all the roles possessed by a user entry.

Roles constitute a LDAP grouping mechanism. A role may have members, which are the entries said to possess the role. Role mechanism enables the following operations:
– enumerating the members of a given role,
– determining whether a given entry possesses a particular role,
– enumerating all the roles possessed by a given entry,

It is further possible to assign a particular role to a given entry and to revoke a particular role from a given entry.

Every role is defined by its own definition entry. A role is uniquely identified by the distinguished name of its definition entry. Role definition entries are LDAP subentries and therefore inherit the subentry mechanism, defined in the ISO/IEC X.509 standard, for scoping. The scope of a role corresponds to the subtree of the role parent entry as illustrated by figure 7. User entries E01 and E02 are in the scope S1 of the role R1 while entry E11 is out of the scope of the role R1. Thus; E01 and E02 are likely to be members

d.

of role R1 while E11 cannot be a member of role R1.

Referring to figure 8, a role can be of "managed" type 801, "filtered" type 802 or "nested" type 803. Each type of role further has two specific object classes 81 that inherit from the *nsRoleDefinition* object class and is related to specific attributes 82 (*nsRoleDN*, *nsRoleFilter*).

On creating a role, members may be assigned to the role as follows:

- members of a managed role have the *nsRoleDN* attribute (or role identifier) in their entry,
- members of a filtered role are entries that match the filter specified in the *nsRoleFilter* attribute (or role filter condition) of the role definition entry,
- members of a nested role are members of the roles specified in the *nsRoleDN* attributes of the nested role definition entry.

Exhibits E2.1, E2.3 and E2.5 respectively contain an example of a managed role, an example of a filtered role and an example of a nested role, in LDIF, according to the prior art. Exhibits E2.2 and E2.4 represent respectively a user entry, possessing the managed role of exhibit E2.1, and a user entry, member of the filtered role of exhibit E2.3.

In the prior art, when a request is made to compute *nsrole* for a given user entry, the directory server tests if the given entry is member of a set of candidate roles. The set of candidate roles may be a list of roles associated with the top suffix of the given entry. This list is prepared in advance in a role cache. The role cache is a data structure updated on creating a new role or on deleting an existing role in the subtree of the top suffix. The role cache contains the list of the roles defined in the subtree of the top suffix. Each role of the role cache is also related to role data, which comprise specific information about the role.

nsrole attribute is computed for an entry depending on a condition (or virtual attribute condition) that comprises a role membership condition, related to *nsRoleDN* attribute and *nsroleFilter* attribute, and a scope condition.

Figure 9a is a flowchart representing the operations performed by the directory server for computing *nsrole* attribute, in order to determine the roles possessed by a given user entry:

a) At operation 200, the directory server receives the request for determining the roles possessed by a given user entry E0;

b) computing *nsrole* attribute starts with operation 202; this operation performs access to the top suffix of entry E0;

c) operation 204 retrieves a role cache associated with the top suffix of E;

d) For each role of the cache role,

   d1) operation 206 retrieves the role data of the current role;

   d2) operation 208 tests if entry E0 possesses the current role, and if so adds the role to the result and select the next role of the list;

e) When all the candidate roles have been tested (operation 210 fails), the directory server assigns the result to *nsrole*, at operation 212.

Reference is now made to the flowchart of figure 9b, representing the different operations performed at operation d2 for checking whether a given user entry E0 possesses a given role R1, according to the prior art.

At the initial operation 50, the directory server receives the request. Operation 51 retrieves role data associated with role R1. These role data may be represented by a data structure comprising specific information about the role, like the type of the role, e.g. "nested", the filter condition when the role is filtered and the role distinguished name *dn*. The role data may be stored in a cache to ease the processing of the request. They are provided from the attributes comprised in the role definition entry.

Operation 52 checks whether entry E0 is in the scope of role R1. This operation may be performed, comparing part of the distinguished names of entry E0 and role R1.

If entry E0 is in the scope of role R1, operation 53 further checks whether role R1 is nested.

If role R1 is not nested, the server operation 54 checks whether entry E0 matches the membership condition of role R1 :

– If role R1 is filtered, the membership condition corresponds to the filter condition

α

identified by *nsRoleFilter* attribute; this filter condition is available in the role data.

- If role R1 is managed, entry E0 should match "nsRoleDN= <role distinguished name>"; the role distinguished name is available in the role data.

If entry E0 matches the membership condition of role R1, at operation 56, the directory server returns that entry E0 possesses role R1.

If role R1 is nested, at operation 55, the directory server recursively performs operations 52 to 56 for each role contained by the nested role. If entry E0 possesses one of the roles contained by role R1 (test 57), at operation 59, the directory server returns that entry E0 possesses role R1.

If entry E0 is not in the scope of role R1 (test 52) or if entry E0 does not match the membership condition of role R1 (test 54), the directory server returns that entry E0 does not possess role R1 (operation 58).

While the operations for determining whether a given entry possesses a particular role and for enumerating all the roles possessed by a given entry are efficiently performed by LDAP server, as illustrated by the flowcharts of figures 9a and 9b, the operations for enumerating the members of a given role is very slow and may return erroneous results. It is due to the fact that these operations require a search based on the role attribute *nsrole*, which is a virtual attribute. For example, to determine the users belonging to the Manager division, a search like "ldapsearch -b dc=sun, dc=com -s subtree (nsrole=cn=managerRole, o=sun.com) dn" may be used, where "cn=managerRole,o=sun.com " is the role of manager users. In the prior art, performing such search would require evaluating the virtual-based filter "nsrole = cn=managerRole, o=sun.com " in the entries of the directory server which is a costly operation.

The invention proposes to index virtual attributes so as to enable such searches to be efficiently performed. According to an aspect of this invention, the directory server comprises a resolving function operable for converting a search filter request based on a virtual attribute (or "virtual" search filter) into a search filter request based on real attributes (or "real" search filter).

The conversion of a virtual search filter into a real search filter is made possible because a virtual attribute is computed for an entry depending on a condition (or virtual attribute condition) which is derived from real attributes stored elsewhere in the directory server. For example, a cos attribute is generated from the real attributes stored in a cos definition entry and in a cos template entry and the role attribute *nsrole* from the real attributes stored in role entries.

The resolving function is part of a filter execution function which is provided by the directory server for evaluating a filter expression. The filter expression may be comprised in a search request from a client.

The general structure of the filter execution function is represented, in figure 10a. The filter execution function 4 transmits a received filter expression to a filter dividing function 40 which is operable for breaking down the filter expression into elementary first filter expressions of the type "attribute operator value".

Each elementary first filter expression is then transmitted to a discriminator 42 capable of determining whether the elementary first filter expression is a real-based one or a virtual-based one.

If the elementary first filter expression is real-based, the discriminator 42 transmits it to a real filter evaluation function 49 which evaluates the elementary first filter expression according to the prior art and provides a corresponding set of entries. The real filter evaluation function 49 may determine the corresponding sets of entries from the indexes.

If the elementary first filter expression is virtual-based, the discriminator 42 transmits it to a virtual filter evaluation function 44 which submits the elementary filter expression to the resolving function 46.

The resolving function 46 converts the first filter expression into a second filter expression based on real attributes, and this second filter expression is sent to the dividing function 40 to be broken down into elementary second filter expressions. Then, each elementary second filter expression is transmitted to the real filter evaluation function 49 for

α

evaluation. The real filter evaluation function 49 provides a set of entries for each one of these elementary second filter expression, the filter execution function 4 being arranged to combine the resulting sets of entries in accordance with the second filter expression provided by the resolving function 46.

The filter execution function 4 then combines the sets of entries obtained for each elementary first filter expressions in accordance with the initial filter expression received by the filter execution function 4.

The embodiments of this invention will be illustrated in the foregoing description through *nsrole* attribute. Understanding the invention starts with a discussion about the real attributes from which *nsrole* is computed.

A user entry would match the filter "nsrole = <roledn>" provided that it possesses the role identified by <roledn>. This implies that the user entry should be in the scope of the role <roledn> and match the membership condition of the role <roledn>.

In the filter expression "nsrole = <roledn>", where <roledn> represents a value of *nsrole*, the following considerations apply, according to the existing role membership conditions:

i) if <roledn> identifies a managed role, the first filter expression "nsrole = <roledn>" corresponds to the second filter expression "nsroledn = <roledn>", where *nsRoleDN* is a real attribute.

ii) if <roledn> identifies a filtered role, the first filter expression "nsrole = <roledn>" corresponds the second filter expression "<roledn.nsRoleFilter>", where <roledn.nsRoleFilter> is the value of *nsroleFilter* attribute of the role identified by <roledn>; the value of *nsroleFilter* attribute is a role filter condition of the type "attribute operator value", where "attribute" is real.

iii) if <roledn> identifies a nested role containing the roles <roledn.nestedRole_1> to <roledn.nestedRole_N>, the first filter expression "nsrole = <roledn>" is equivalent to the union of the filter expressions "nsrole = <roledn.nestedRole_k>" (with k from 1 and N). Each filter expression "nsrole = <roledn.nestedRole_k>" corresponds to a second filter expression according to operation i) or ii), the nested role identified by <roledn> being associated with the union of the second filter expressions found for

α

the contained roles.

For example, in reference to the roles defined in Exhibit E2:

i1) "nsrole=cn=Marketing,ou=people,dc=example,dc=com" (Exhibit E2.1) corresponds to "nsRoleDN = cn=Marketing,ou=people,dc=example,dc=com",

ii1) "nsrole=cn=SalesFilter,ou=people,dc=example,dc=com" (Exhibit E2.3) corresponds to "o=sales",

iii1) "nsrole=cn=MarketingSales,ou=people,dc=example,dc=com" (Exhibit E2.5) this nested role contains:

- the role "nsrole = cn=Marketing,ou=people,dc=example,dc=com" corresponding to " nsRoleDN = cn=Marketing,ou=people,dc=example,dc=com", and
- the role "nsrole = cn=SalesFilter,ou=people,dc=example,dc=com" corresponding to "o=sales",

and therefore the nested role corresponds to the following real filter expression: (|(nsRoleDN=cn=Marketing,ou=people,dc=example,dc=com)(o=sales)).

Figure 10b is a flowchart representing the operations performed by the directory server for indexing *nsrole* attribute, in response to a search request comprising an equality search filter of the type:" nsrole =<roledn>".

On receiving a search request, the directory server previously performs operations 102 to 105 of figure 6. Like in the prior art, operation 105 determines if the attribute associated with the elementary search filter is indexed. However, if the attribute is not indexed, the directory server further determines whether the elementary search filter is based on a virtual attribute *nsrole* (operation 500).

If the elementary search filter is based on this virtual attribute, e.g. "nsrole =<roledn>", operation 502 determines the type of the role identified by <roledn>. This may be done by accessing the suffix of the role and getting the role data of the role cache associated with that suffix:

- If the role is "managed", operation 504 converts the virtual search filter expression "nsrole = <roledn>" into the corresponding real search filter "nsroledn= <roledn>";
- If the role is "filtered", operation 506 converts the virtual search filter expression

"nsrole = <roledn>" into the corresponding real search filter expression "<roledn.nsroleFilter>", where <roledn.nsroleFilter> is the value of *nsRoleFilter* attribute of the role identified by <roledn>;

– If the role is "nested", the directory server recursively performs operations 502 to 506 for each contained role (operation 508), and the "real" search filter expression corresponding to the "virtual" search filter expression "nsrole = <roledn>" is attached the union of the "real" search filters expressions found for the contained roles (operation 510).

Operation 512 then evaluates the real search filter provided by any of operations 504, 506 or 510 according to operations 104 to 115 of figure 6. For an optimal search, equality indexes may have been previously created for the real attributes *nsRoleDN* and for the filters defined by the filtered roles (e.g. o=sales).

Operation 512 provides a list of candidates entries representing the entries matching the real search filter.

However, the list of candidate entries provided by operation 512 may contain more entries than required. The conversion of the virtual search filter "nsrole=<roledn>" into a "real" search filter does not take the scope restriction into account. Indeed, some of the entries matching the "real" search filter associated with the "virtual" search filter may not be in the scope of the role identified by <roledn>. For example, in figure 11, the user entry E02 and the user entry E11 comprise " nsRoleDN = R1", R1 being a managed role. Operation 512 would return entry E02 and entry E11, in response to a search request based on the filter (nsrole=R1). But, entry E02 is in the scope S1 of role R1 while entry E11 is not in the scope S1 of role R1, and therefore, entry E02 possesses R1 (E02 matches "nsrole=R1") and entry E11 does not actually possess role R1.

Operation 514 is performed to restrict the set of entries provided by operation 512 to those which match the role scope condition. Operation 514 evaluates the initial virtual search filter restricted for the list of the candidate entries provided by operation 512. The virtual search filter can now be applied efficiently as the number of candidate entries is not important. For example, to evaluate this virtual search filter, the directory server may

α

compute *nsrole* attribute for each entry of the list of candidate entries to check if the entry is actually member of the role.

Additionally, operation 500 previously determines if the filter comprises a negation, e.g., (!(nsrole = <roledn>). Such negative search filters are not currently indexed. It is due to the fact that search filters that comprise a negation of the type (!(attr=<attr_value>)) not only returns the entries where *attr* has a value different from <attr_value>, but also entries where the *attr* attribute is not defined. As well, the invention does not support negative filters. If a filter of the type (!(nsrole = <roledn>) is detected, the search filter is resolved according to the prior art (operation 501).

The embodiments of this invention have been described in reference to an equality search filter of the type (nsrole=<roledn>). Alternatively, the invention may process other type of search filters. For example, the request may comprise the presence search filter "nsrole=*", which means that all the entries containing one or more values of *nsrole* are searched. The presence index may be maintained from the equality index. The directory server may directly determines the list of candidate entries, at operation 504, 506 or 508 of figure 10b by performing the union of the entries stored in the equality indexes.

It will be appreciated that many of the features being described in terms of operations may also be encompassed in terms of functions in a directory server and/or directory sevrer system, and conversely.

This invention also encompasses software code, especially when made available on any appropriate computer-readable medium. The expression "computer-readable medium" includes a storage medium such as magnetic or optic, as well as a transmission medium such as a digital or analog signal. Such software code may include data and/or metadata.

This invention also encompasses the software code to be added to existing directory server functionalities to perform anyone of the various new functionalities, as described above, which may be used independently of each other.

On another hand, a number of features have been positively described, using absolute language, to help understanding the LDAP example. Each such feature should be considered as exemplary only, and is not intended to restrict the scope of this invention in any way.

Exhibit E1 - search filters

E1.1- Parameters of *ldapsearch* function

ldapsearch [*options*] [*search filter*] [*list of attributes*]

E1.2- main options

Option	Description
-b	Specifies the starting point for the search in the DIT. The value specified is a distinguished name that currently exists in the database. For example : -b "cn=bob, dc=siroe, ou=people, dc=siroe, dc=com"
-D	Specifies the distinguished name used to authenticate to the server. The value specified must be a DN recognized by the Directory Server and must be allowed to search for the entries. For example : "-D uid=bjensen, dc=siroe,dc=com"
-h	Specifies the machine hostname or machine IP address of the Directory Server. For example : "-h mozilla"
-l	Specifies the maximum time (expressed in seconds) to take for a search request. For example : "-l 500 "
-p	Specifies the TCP port number used by the Directory Server. For example : "-p 1049"
-s	Specifies the search scope. The scope may be : - the entry specified in "-b option " (*base-Search*) ; - the immediate children of the entry specified in the "- b option "(*one-Search*). The entry specified in the -b option is not searched. - the entry specified in the "-b option " and all of its descendants (*subSearch*).
-w	Specifies the password associated with the distinguished name that is specified in the -D option. For example :"-w secret "
-x	Specifies that the search results are transmitted to the server rather than on the client. In general, it is faster to transmit results to the server.
-z	Specifies the maximum number of entries to return in response to a search request. For example :"-z 500"

E1.3- Search filter types

Search type	**operator**	**description**
Equality	=	Returns entries that contain attribute values exactly matching the value specified by the filter. For example: "sn=jensen"
Substring	**= string* string**	Returns entries that contain attributes comprising the substring specified by the filter. For example : "cn=joe*", "cn=*Jensen ", "cn=*Joe*", "cn=B*Joe" * indicates zero or more characters
Greater than or equal to	**>=**	Returns entries that contain attributes that are greater than or equal to the value specified by the filter. For example : " age >= 40"
Less than or equal to	**<=**	Returns entries that contain attributes that are less than or equal to the value specified by the filter. For example : "age <= 40"
Presence	**=***	Returns entries that contain one or more values for the attribute specified by the filter. For example :"cn=*", " telephonenumber=*"
Approximate	**~=**	Returns entries that contain the specified attribute with a value that is approximately equal to the value specified in the search filter. For example : "sn~=janson" could return "sn=jensen"

E1.4- Example of complex search filter

```
" (!cn=Bob Jones*)((cn=Bob*)| (cn=*Jones))(age<=40) "
```

E.1.5- boolean operators

Operator	Symbol	Description
AND	&	All specified filters must be true for the statement to be true. For example, (&(filter)(filter)(filter)...)
OR	|	At least one specified filter must be true for the statement to be true. For example, (|(filter)(filter)(filter)...)
NOT	!	The specified statement must not be true for the statement to be true. Only one filter is affected by the NOT operator. For example, (!(filter))

α

Exhibit E2-Role definition entries

E2.1 - Managed role

```
dn : cn = Marketing, ou = people, dc = example, dc = com
objectclass : top
objectclass : LDAPsubentry
objectclass : nsRoleDefinition
objectclass : nsSimpleRoleDefinition
objectclass : nsComplexRoleDefinition
cn : Marketing
description : managed role for marketing staff
```

E2.2 - Entry member of Marketing role

```
dn : cn = Joe, ou = people, dc = example, dc = com
objectclass : person
cn : Joe
sn : Bradford
userpassword : joepasswd
nsRoleDN : cn = Marketing, ou = people, dc = example, dc = com
```

E2.3- Filtered role

```
dn : cn = SalesFilter, ou = people, dc = example, dc = com
objectclass : top
objectclass : LDAPsubentry
objectclass : nsRoleDefinition
objectclass : nsComplexRoleDefinition
objectclass : nsFilteredRoleDefinition
cn : SalesFilter
nsRoleFilter : o= sales
description : filtered role for sales staff
```

E2.4- Entry member of Filtered role

```
dn : cn = Richard, ou = people, dc = example, dc = com
objectclass : person
cn : Richard
sn : Parker
userpassword : richardpasswd
o : sales
```

E2.5- Nested role

```
dn : cn = MarketingSales, ou = people, dc = example, dc = com
objectclass : top
objectclass : LDAPsubentry
objectclass : nsRoleDefinition
objectclass : nsComplexRoleDefinition
objectclass : nsNestedRoleDefinition
cn : MarketingSales
nsRoleDN : cn = Marketing, ou = people, dc = example, dc = com
nsRoleDN : cn = SalesFilter, ou = people, dc = example, dc = com
description : nested role for marketing and sales staff
```

Claims

1. A directory server, capable of interacting with entries organized in a tree structure, each entry having attributes, said attributes comprising real attributes each having a value stored in the entry,

the directory server comprising:

- a mechanism capable of associating a virtual attribute to an entry, subject to a virtual attribute condition being verified, the virtual attribute condition being derived from data located elsewhere in the tree structure, and
- a resolving function, capable of receiving a first filter expression, based on a virtual attribute, for converting it into one or more second filter expressions, containing real attributes, and being computed from the first filter expression and from the virtual attribute condition.

2. The directory server of claim 1, further comprising indexing tables, each related to a real attribute, and having an ordered list of attribute values with corresponding entry identifiers, said indexing tables being extendable to those of the real attributes which may be contained in the second filter expressions.

3. The directory server of claim 2, wherein some of the indexing tables are cached.

4. The directory server as claimed in claim 2 or 3, further comprising a filter execution function having :

- a real filter evaluation function, capable of receiving a real-based filter expression, based on a real attribute, for determining a corresponding set of entries,
- a virtual filter evaluation function, capable of receiving a virtual-based filter expression, based on a virtual attribute, for submitting it to the resolving function, as a first filter expression, and for subsequently submitting resulting second filter expressions to the real filter evaluation function, and
- a discriminator, for determining whether a filter expression is a real-based one or a virtual-based one.

5. The directory server of claim 4, wherein the real filter evaluation function is arranged

α

to determine the set of entries from an indexing table for the real attribute, if available.

6. The directory server of claim 4 or 5, further comprising a filter manager, responsive to the receipt of a request comprising a filter expression for calling the filter execution function and evaluating the request in the set of entries determined by the filter execution function.

7. The directory server as claimed in any of claims 4 through 6, wherein the filter manager has a filter dividing function, capable of splitting an input filter expression into elementary filter expressions.

8. The directory server of claim 7, wherein the filter dividing function is operable to split a filter expression presented as an input of the filter execution function into elementary filter expressions for application to the filter execution function, and the filter execution function is arranged to combine the respective results in accordance with the combination of the elementary filter expressions in the input filter expression.

9. The directory server of claim 7, wherein the filter dividing function is operable to split second filter expressions into elementary second filter expressions for application to the real filter evaluation function, and the filter execution function is arranged to combine the respective results in accordance with the combination of the second elementary filter expressions.

10. The directory server as claimed in any of claims 1 through 9, wherein the first filter expression comprises an operator intervening between a virtual attribute name and a virtual attribute value.

11. The directory server as claimed in any of claims 1 through 8, wherein the virtual attribute is a role attribute, related to a role entry in the tree structure, and the virtual attribute condition comprises a role membership condition, the resolving function being operable for converting a first expression comprising the virtual role attribute into a second filter expression based on said role membership condition.

12. The directory server of claim 11, in which a user entry meets the role membership condition if it has a real attribute designating a role identifier, attached to the role entry.

13. The directory server of claim 11, in which a user entry meets the role membership condition if it meets a role filter condition, attached to the role entry.

14. The directory server as claimed in claim 12 or 13, in which the role entry contains a plurality of roles, wherein the resolving function is operable for repetitively receiving a filter expression comprising the role attribute with a value identifying individual ones of such contained roles.

15. The directory server as claimed in any of claims 11 through 14, further comprising a filter execution function capable of determining a set of entries from said second filter expression, and of restricting that set of entries to those which verify said virtual attribute condition.

16. The directory server as claimed in any of claims 11 through 15, in which a role entry has a scope in the tree structure, wherein the virtual attribute condition comprises a scope condition.

17. A method of operating a directory server system, comprising a directory server interacting with entries organized in a tree structure,

each entry having attributes, said attributes comprising real attributes, each real attribute having a value stored in the entry, and virtual attributes, each virtual attribute being associated to an entry, subject to a virtual attribute condition being verified, the virtual attribute condition being derived from data located elsewhere in the tree structure,

the method comprising the steps of :

a) receiving a first filter expression,

b) if the first filter expression is based on a virtual attribute, converting said first filter expression into one or more second filter expressions, containing real attributes, said one or more second filter expressions being computed from the first filter expression and from the virtual attribute condition.

18. The method of claim 17, further comprising the step of constructing indexing tables, each related to a real attribute, and having an ordered list of attribute values with corresponding entry identifiers, said indexing tables being extendable to those of the real attributes which may be contained in the second filter expressions.

n (31 pages) CABINET NETTER

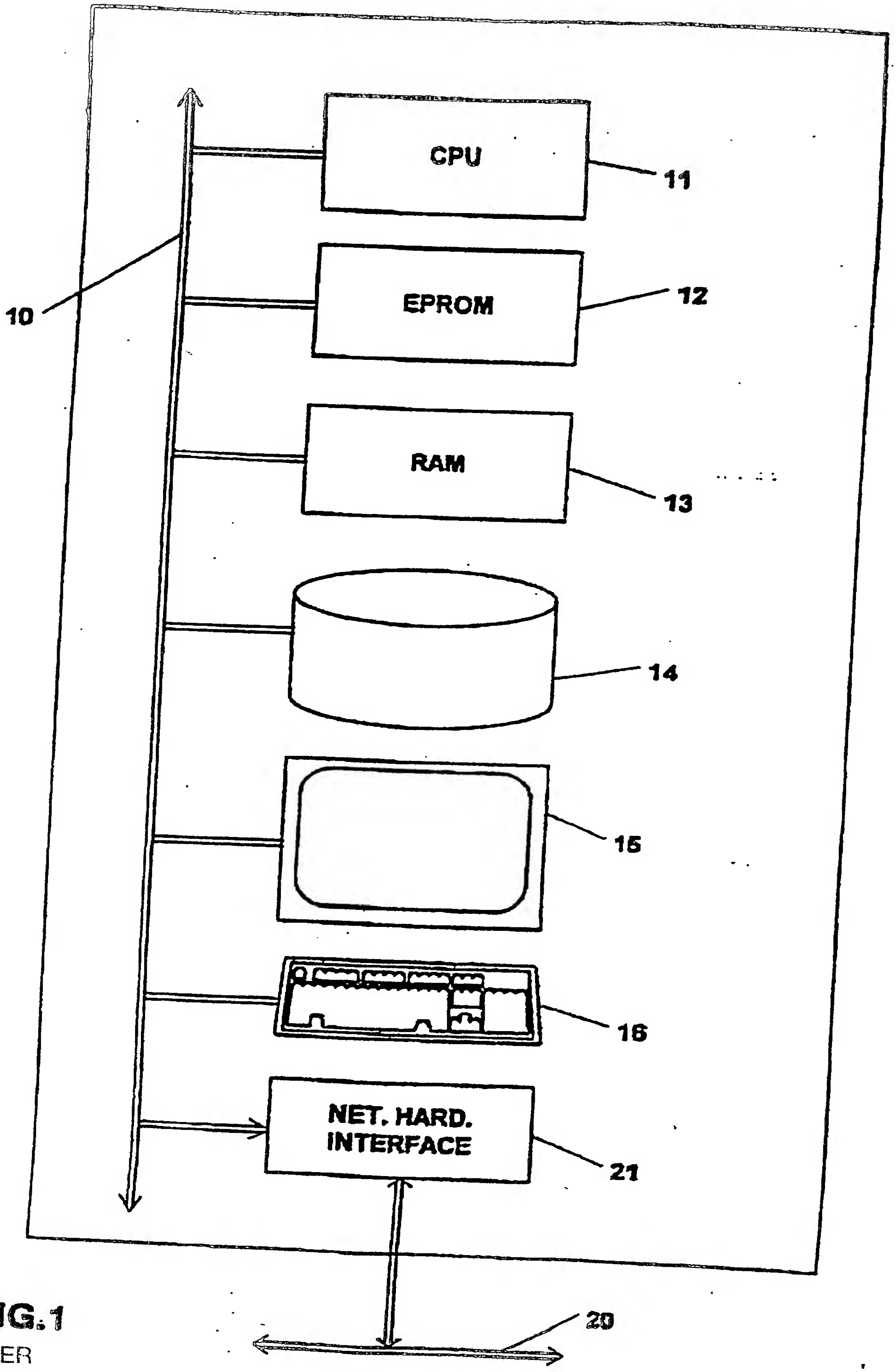


FIG.1

PRIOR ART

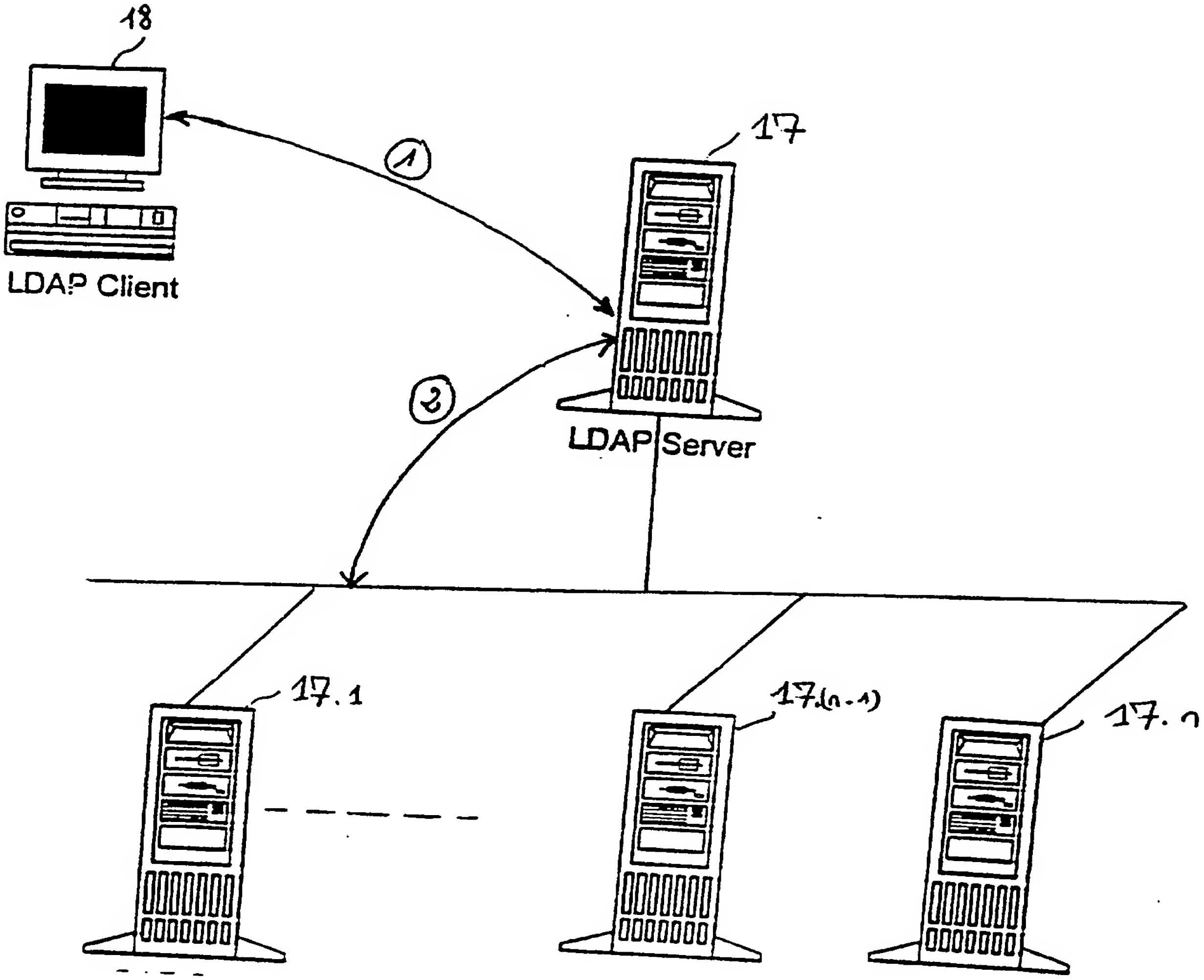


FIG. 2

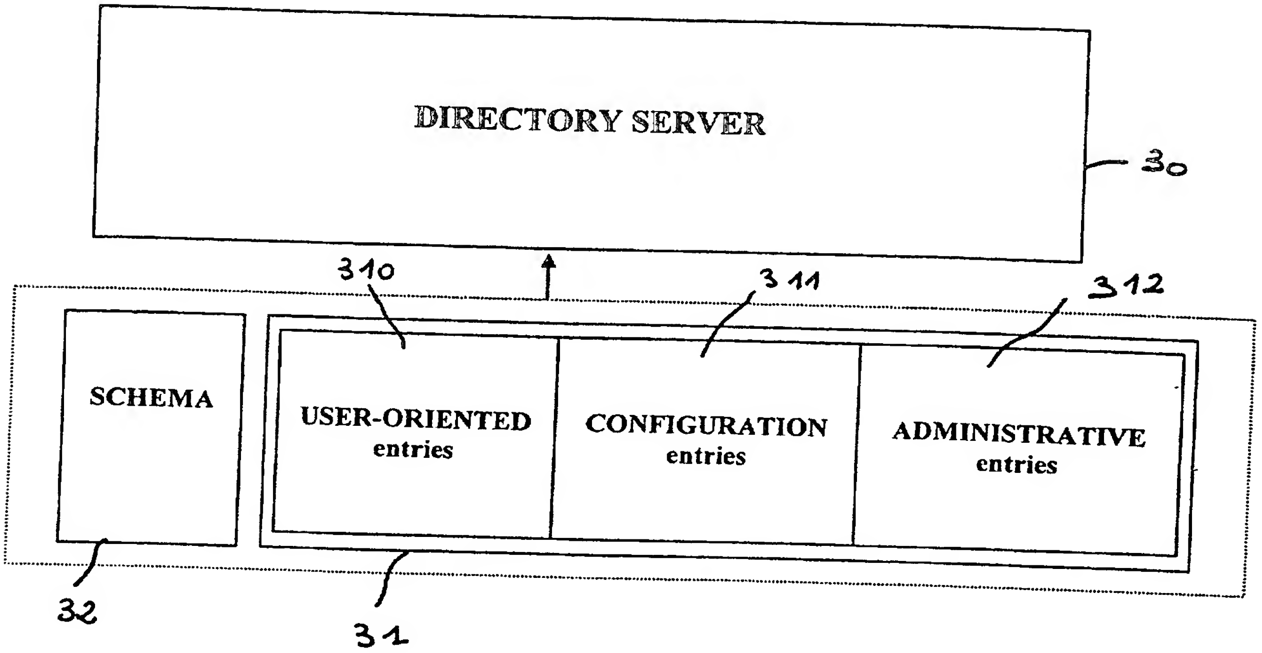

DIRECTORY SERVER
30
310
311
312
SCHEMA
USER-ORIENTED entries
CONFIGURATION entries
ADMINISTRATIVE entries
32
31

FIG 3

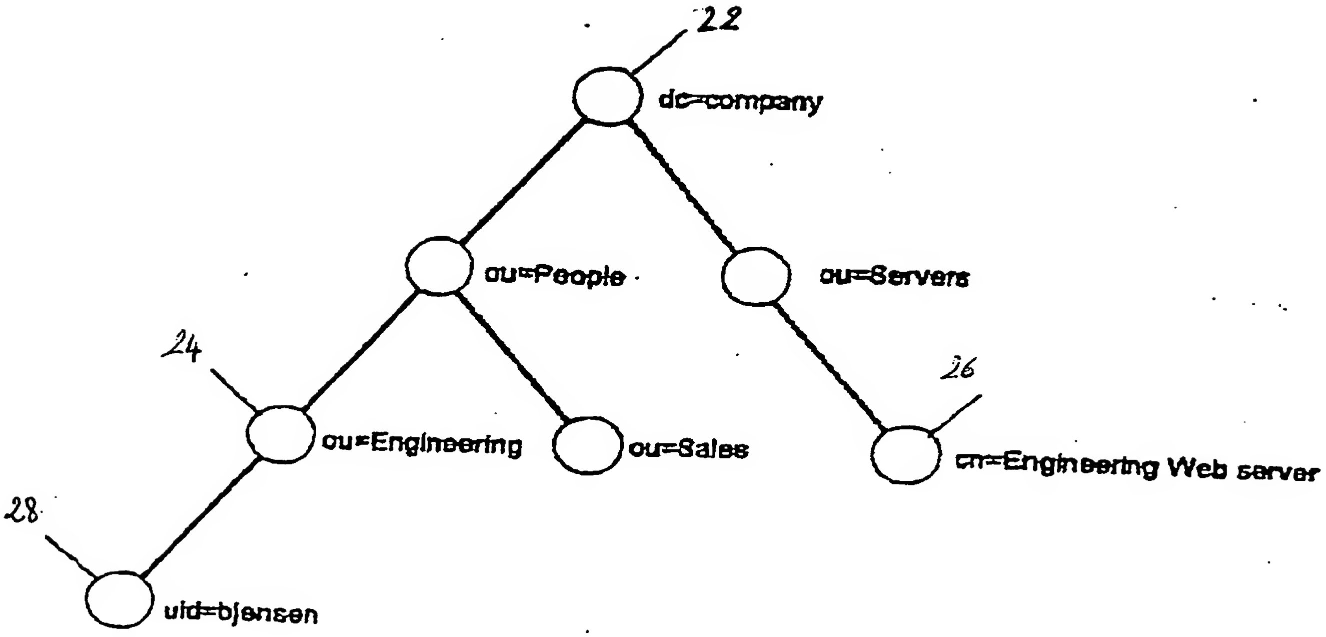
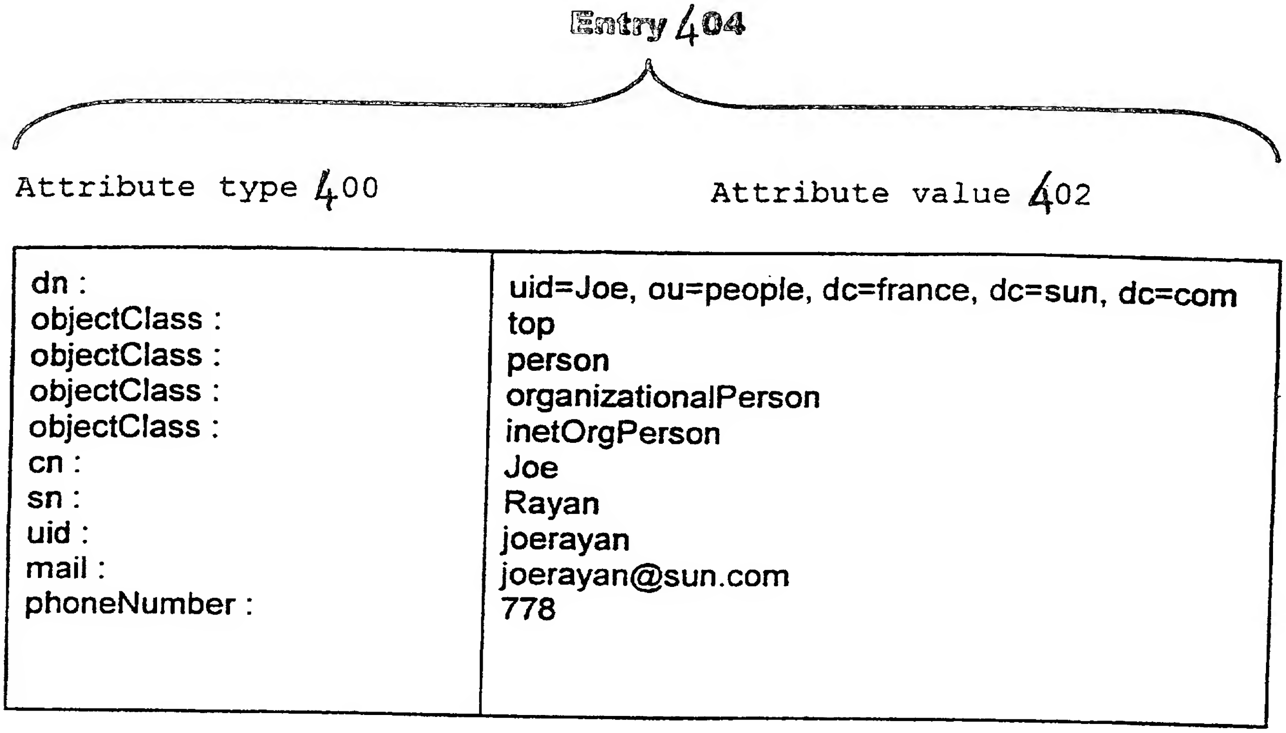


FIG 4

(PRIOR ART)



dn :	uid=Joe, ou=people, dc=france, dc=sun, dc=com
objectClass :	top
objectClass :	person
objectClass :	organizationalPerson
objectClass :	inetOrgPerson
cn :	Joe
sn :	Rayan
uid :	joerayan
mail :	joerayan@sun.com
phoneNumber :	778

Fig 5.

(PRIOR ART)

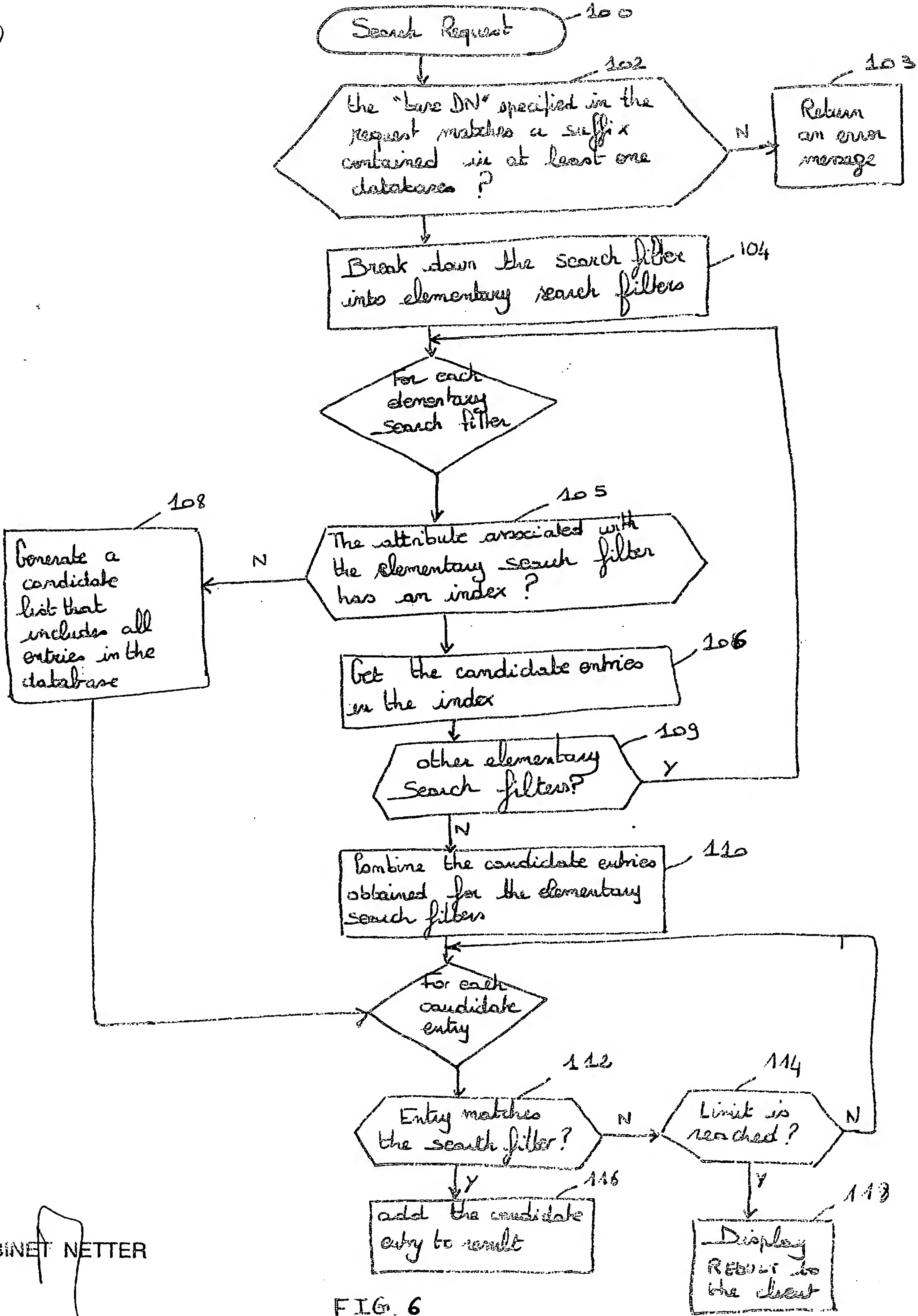

Search Request
100
the "base DN" specified in the request matches a suffix contained in at least one database ?
102
N
103
Return an error message
Break down the search filter into elementary search filters
104
For each elementary search filter
105
The attribute associated with the elementary search filter has an index ?
N
108
Generate a candidate list that includes all entries in the database
106
Get the candidate entries in the index
109
other elementary search filters?
Y
N
110
Combine the candidate entries obtained for the elementary search filters
For each candidate entry
112
Entry matches the search filter?
N
114
Limit is reached?
N
Y
Y
116
add the candidate entry to result
118
Display RESULT to the client

FIG. 6

CABINET NETTER

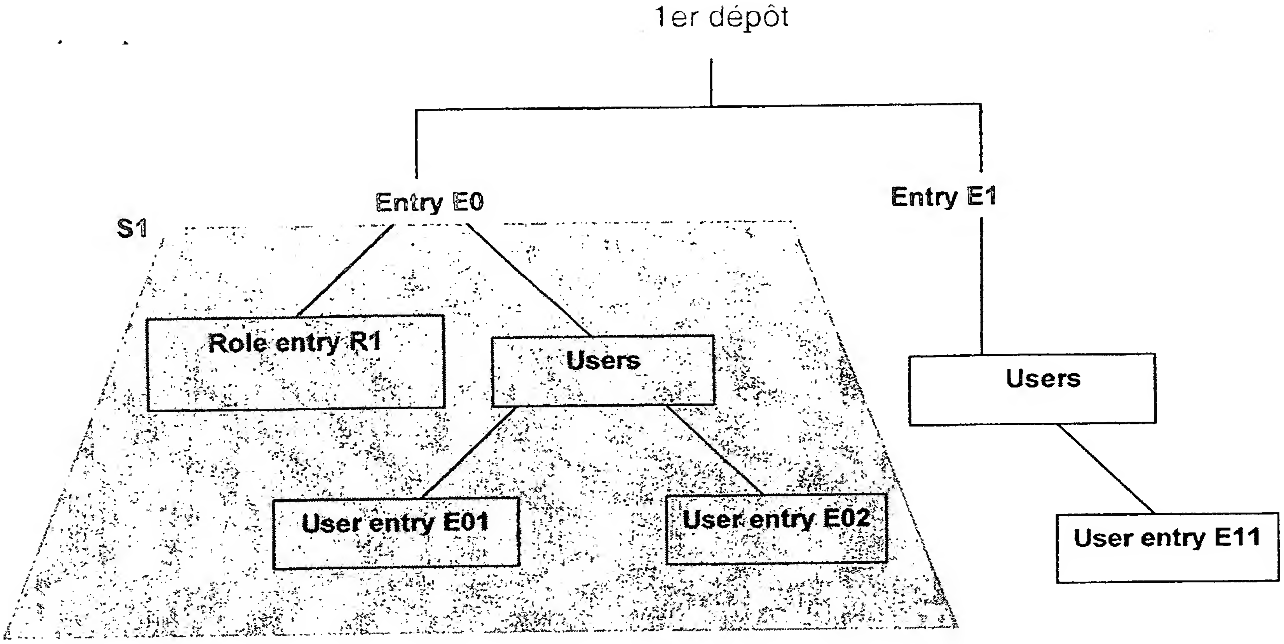

1er dépôt
Entry E0
Entry E1
S1
Role entry R1
Users
Users
User entry E01
User entry E02
User entry E11

FIG 7

81 82

	Role Type	Object Classes	Attributes
801	**Managed Role**	nsSimpleRoleDefinition nsManagedRoleDefinition	description (optional)
802	**Filtered Role**	nsComplexRoleDefinition nsFilteredRoleDefinition	NsRoleDN description (optional)
803	**Nested Role**	nsComplexRoleDefinition nsNestedRoleDefinition	NsRoleDN description (optional)

Figure 8

(PRIOR ART)

CABINET NETTER

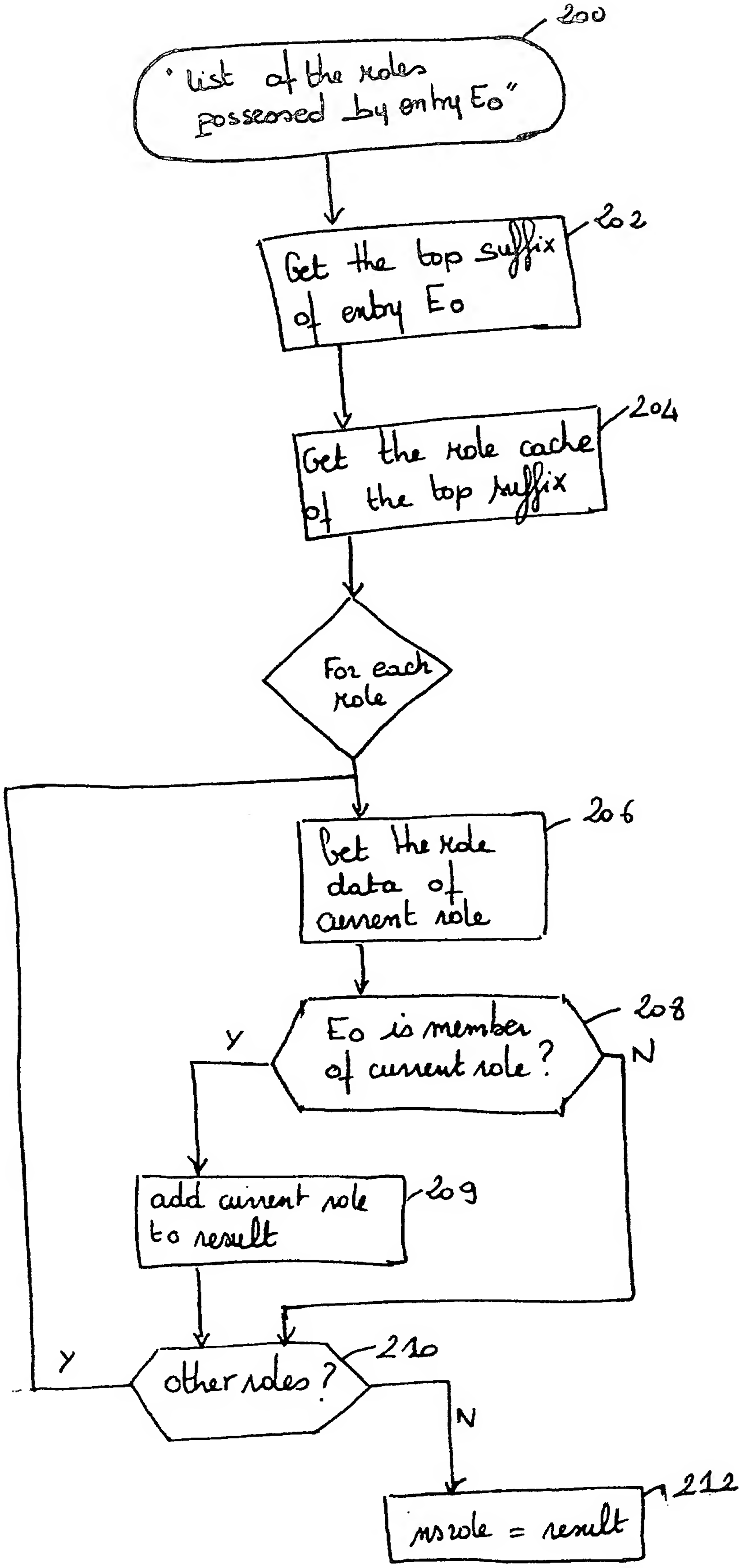

200
"list of the roles possessed by entry Eo"
Get the top suffix of entry Eo
202
Get the role cache of the top suffix
204
For each role
Get the role data of current role
206
Eo is member of current role ?
208
Y
N
add current role to result
209
other roles ?
210
Y
N
nsrole = result
212

FIG 9 a

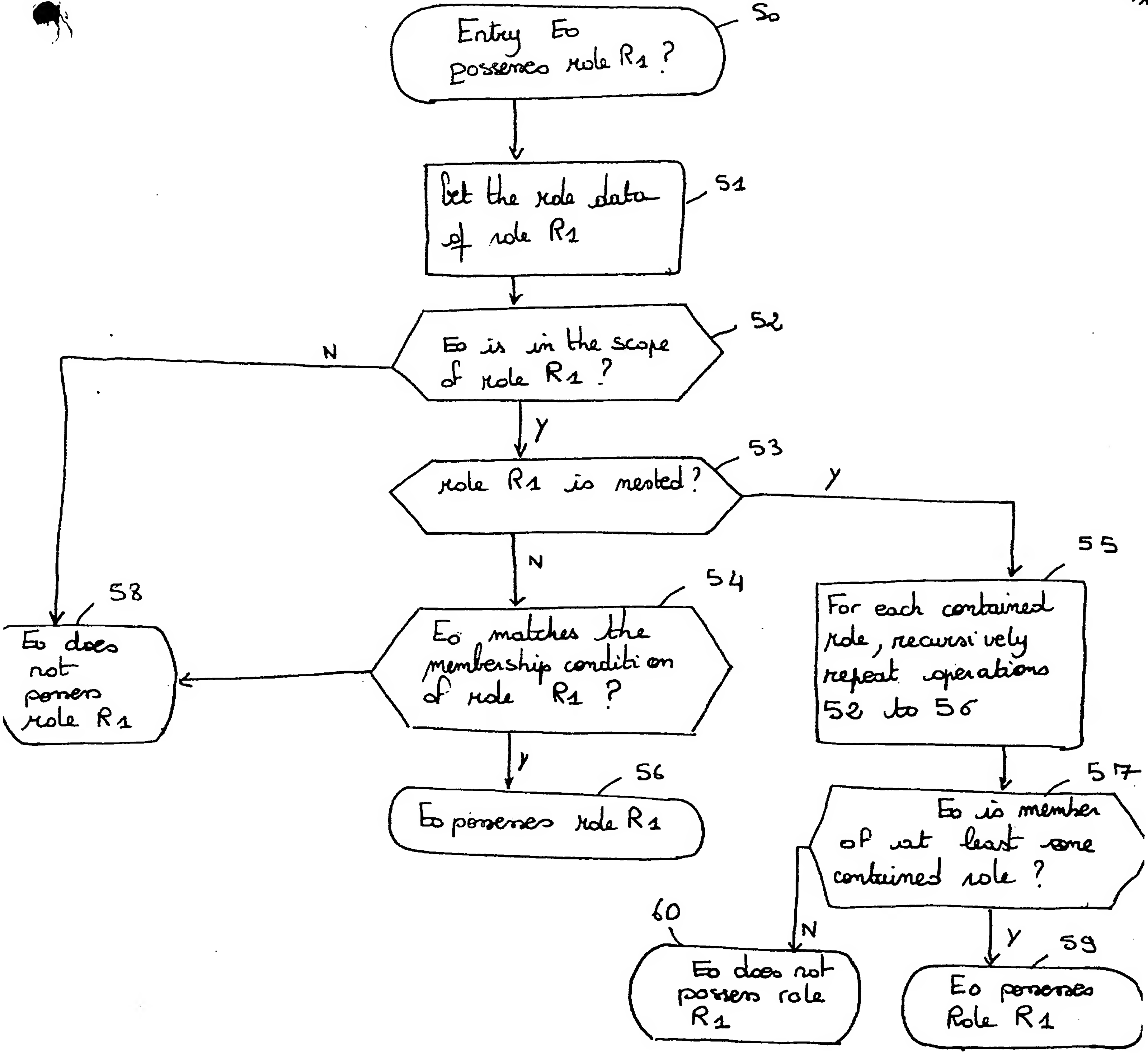

S0
Entry E0 possesses role R1 ?
S1
Get the role data of role R1
S2
E0 is in the scope of role R1 ?
N
Y
S3
role R1 is nested ?
Y
N
S5
For each contained role, recursively repeat operations S2 to S6
S8
E0 does not posses role R1
S4
E0 matches the membership condition of role R1 ?
Y
S6
E0 possesses role R1
S7
E0 is member of at least one contained role ?
60
N
Y
S9
E0 does not posses role R1
E0 possesses Role R1

FIG 9b

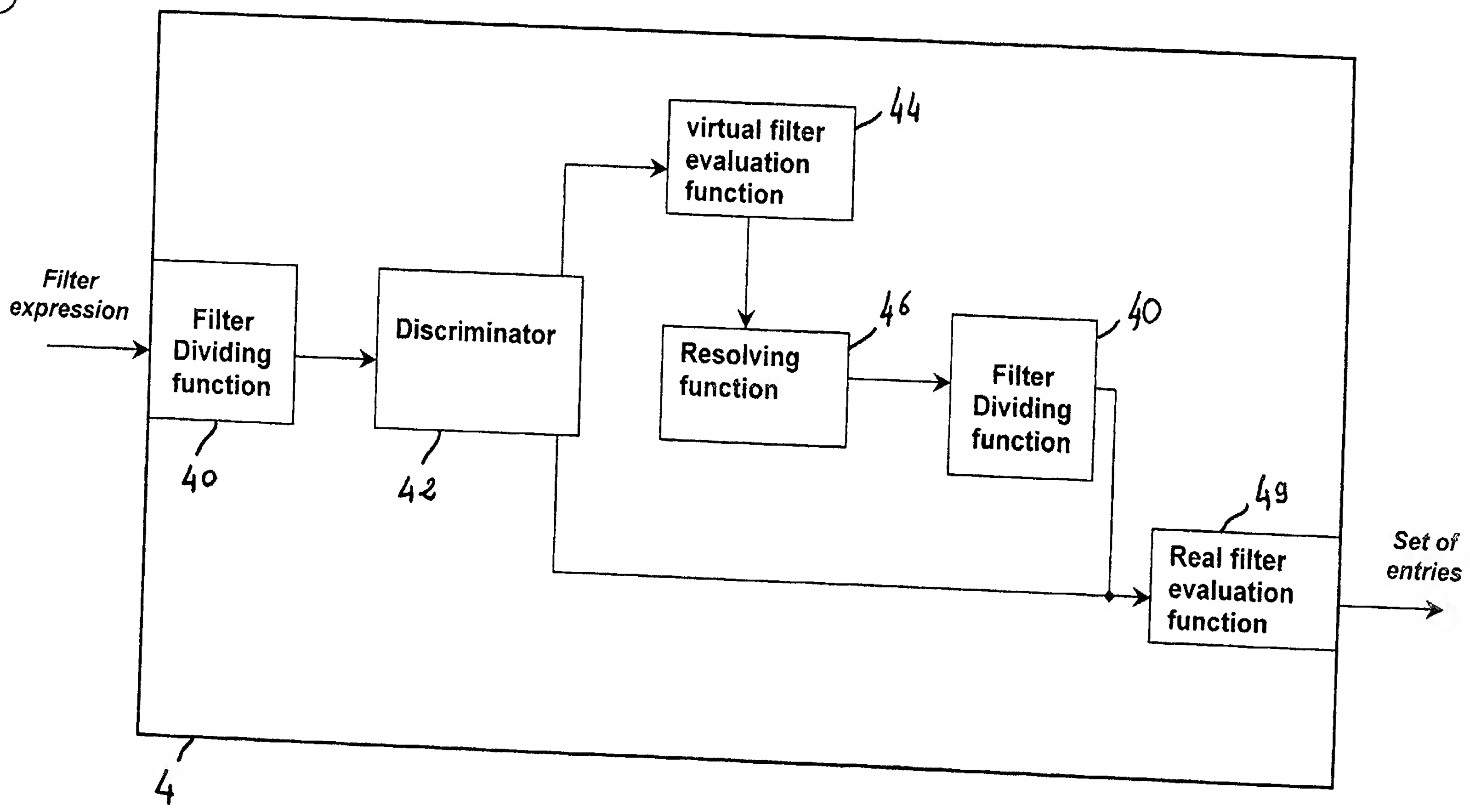

Filter expression
Filter Dividing function
40
Discriminator
42
virtual filter evaluation function
44
Resolving function
46
Filter Dividing function
40
Real filter evaluation function
49
Set of entries
4

Figure 10 a

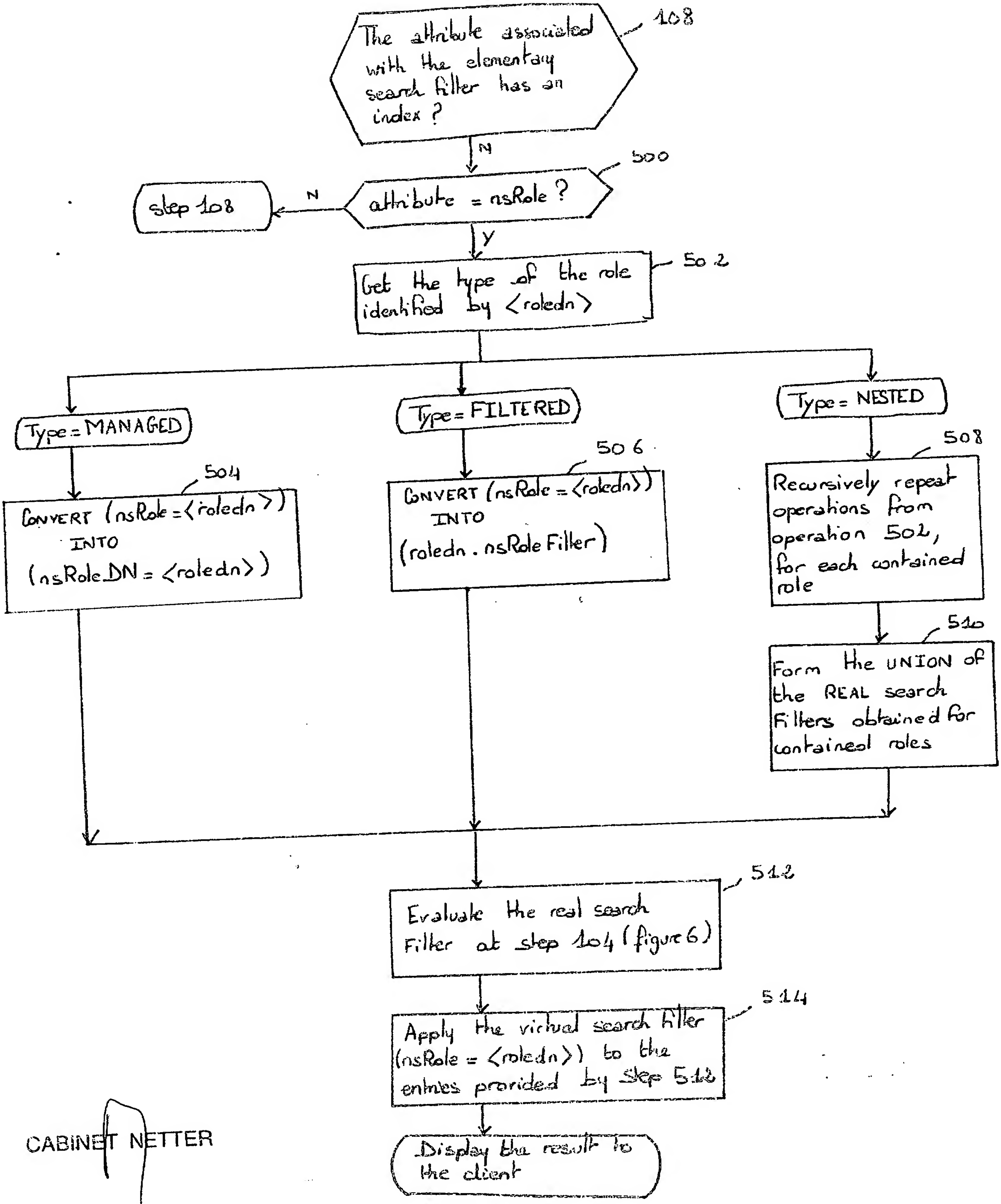

The attribute associated with the elementary search filter has an index ?
108
N
500
step 108
N
attribute = nsRole ?
Y
Get the type of the role identified by <roledn>
50 2
Type = MANAGED
Type = FILTERED
Type = NESTED
504
CONVERT (nsRole = <roledn>) INTO (nsRoleDN = <roledn>)
50 6
CONVERT (nsRole = <roledn>) INTO (roledn . nsRole Filter)
508
Recursively repeat operations from operation 502, for each contained role
510
Form the UNION of the REAL search Filters obtained for contained roles
512
Evaluate the real search Filter at step 104 (figure 6)
514
Apply the virtual search filter (nsRole = <roledn>) to the entries provided by step 512
Display the result to the client

FIG 10b

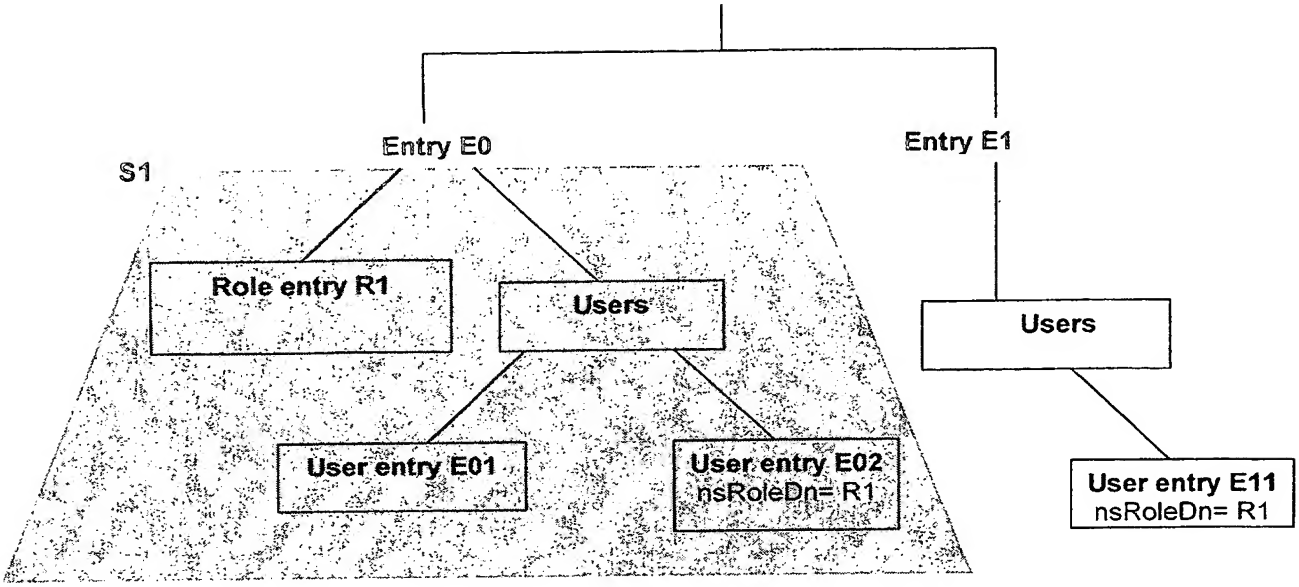


FIG. 11

CABINET NETTER